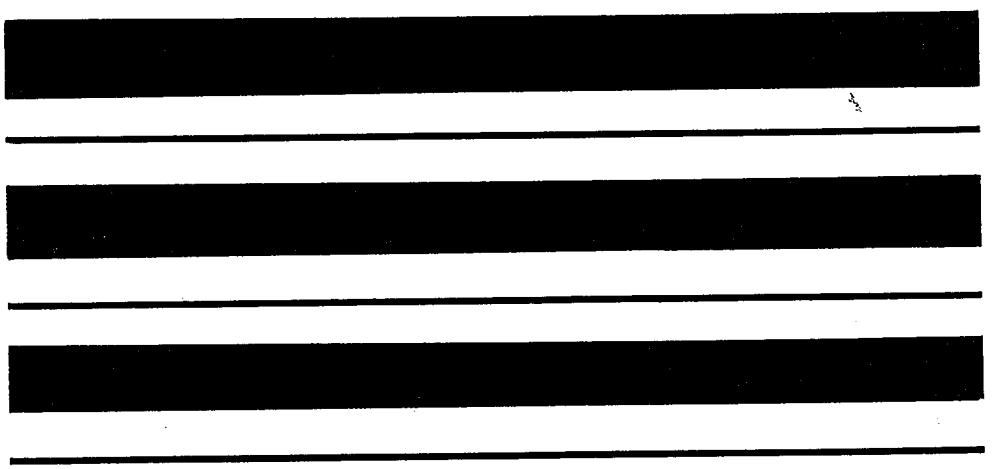
An Artificial Intelligence Approach to VLSI Design

Thaddeus J. Kowalski



Kluwer Academic Publishers

Distributors for North America:
Kluwer Academic Publishers
101 Philip Drive
Assinippi Park
Norwell, MA 02066, USA

Distributors outside North America:
Kluwer Academic Publishers Group
Distribution Centre
P.O. Box 322
3300 AH Dordrecht
THE NETHERLANDS

Library of Congress Cataloging in Publication Data

Kowalski, Thaddeus J.
An artificial intelligence approach to VLSI design.

(The Kluwer international series in engineering and computer science; SECS 4)
Bibliography: p.
Includes index.
1. Integrated circuits — Very large scale integration.
2. Artificial intelligence. 3. Expert systems.
4. Digital electronics. I. Title. II. Series.
TK7874.K675 1985 621.3819'5835 85-7581
ISBN 0-89838-169-X

Fourth printing, 1988

Printed in the United States of America

Chapter 2

KNOWLEDGE-BASED EXPERT SYSTEMS

During the past decade KBESs have been developed by researchers in artificial intelligence to help solve problems whose structures do not lend themselves to recipe-like solutions. These systems differ from previous efforts in problem solving by effectively coping with the enormous search spaces of alternatives found in real-world problems. The key to success in KBESs is the ability to use domain-rich knowledge to recognize familiar patterns in the current problem state and act appropriately. This tool is based on the premise that humans solve problems by recognizing one of many familiar patterns in the current situation and applying the appropriate actions when this pattern occurs. Their recognition of the pattern is not based on the complete current situation, nor is it recognized with absolute certainty. The presence and absence of patterns can be used to help establish and rule out actions for a given situation.

Researchers have developed many KBESs whose features[35] have matured and grown into usable engineering tools.[36,37,38] These systems exploit special knowledge to solve difficult problems in many domains. Examples are: DENDRAL, mass-spectrum analysis; MYCIN, medical diagnosis; PROSPECTOR, mineral exploration; R1, VAX computer configuration; to name only a few. This chapter introduces general

components of KBESs, describes their general features, describes two implementations, and provides references to several other systems. Chapter 3 discusses how the KBES approach has been used to develop an intelligent CAD tool, DAA.

2.1 General Components of a Knowledge-Based Expert System

The problem domains and features of the existing KBESs differ widely, but many have three components in common: a working memory, a rule memory, and a rule interpreter.

2.1.1 The working-memory component. The working memory is a collection of attribute-value pairs that describe the current situation. They resemble the data structures in conventional programming languages:

```
struct foo {
        <attribute 1> = <value 1>;
        ...
        <attribute n> = <value n>;
};
```

Some systems also represent goals and sub-goals as named attribute-value pairs in working memory.

2.1.2 The rule memory component. The rule memory is a collection of conditional statements that operate on elements stored in the working memory. The statements resemble the conditional statements of conventional programming languages:

```
IF:
        <antecedent 1>
        ...
        <antecedent n>

THEN:
        <consequence 1>
        ...
        <consequence m>
```

The rule memory is a collection of knowledge *chunks* about a particular problem domain. Most rule-based systems contain hundreds of rules that have been *painfully* extracted from months of interviewing experts. Acquiring knowledge from experts is difficult because although

they are skillful at doing the task, they are not effective at explaining precisely how they do the task. To give a feel for the number of rules: MYCIN has about four-hundred fifty rules, R1 has about eight-hundred fifty rules (the new version XCON has three-thousand three-hundred rules),[39] and PROSPECTOR has about one-thousand six hundred rules. Systems like MYCIN have added debugged rules at an average rate of about two per week. This in no way accounts for the hundreds of rules that came and went in the debugging of the rule memory.

2.1.3 The rule interpreter component. The rule interpreter pattern matches the working-memory elements against the rule memory to decide what rules apply to the given situation. Some tool-kits allow a degree of certainty to be associated with each consequence that suggests the degree to which the consequence follows from the antecedents. Others allow a pair of certainties to be associated with each consequence that suggests both how sufficient the presence of the antecedents are for establishing the consequence and how sufficient the absence of the antecedents are for not establishing the consequence. Still others apply the consequences with absolute certainty if the antecedents are present in working memory. The selection process of rules can be data driven, goal driven, or a combination of data and goal driven.

A data-driven selection process looks through the rule memory for a rule whose antecedents match elements in the working memory. This is also called forward-chaining or antecedent reasoning. The consequences of the rule are applied, and the process is repeated until no more rules apply or until a rule explicitly stops the process.

A goal-driven selection process looks through the rule memory for a rule whose consequences can achieve the current goal. This is also called backward-chaining or consequent reasoning. If the antecedents of this rule match the elements in working memory, then the consequences are applied and the goal is satisfied. If the antecedents of this rule are not all present in working memory, then the missing antecedent replaces the current goal and the process is repeated until either all the sub-goals are satisfied or no more rules are applicable. If no more rules are applicable, the user can be queried for missing information to place in working memory. This backward-chaining reason also facilitates explanations of how the system had reached a particular conclusion and why it needed certain pieces of knowledge.

2.2 General Features of a Knowledge-Based Expert System

Separating expert knowledge from the reasoning mechanism implies several general features common to knowledge-based expert systems. The knowledge engineer can incrementally add new rules and refine old ones because the rules are chunks of domain knowledge and have minimal interaction with other rules in the rule memory. The rule interpreter can be queried about *why* it needs an additional piece of information and *how* it solved a problem by describing the goals and the rules it has used to solve a problem. A rule interpreter can be developed for aiding the acquisition of knowledge,[40] by checking its own rule memory for oddity, consistency and omitted areas. A rule interpreter can also be developed for aiding the instruction of knowledge,[41] by trying to determine what knowledge students seem to possess and how that knowledge corresponds to its rule memory. Finally, the same rule interpreter may be used by many rule memories and working memories to develop KBESs for different problem domains.

2.3 Example Knowledge-Based Expert Systems

During the past decade many KBESs have been developed for such purposes as interpretation, diagnosis, monitoring, prediction, planning and design. This section discusses two examples — a medical diagnostic system, MYCIN, and a system used to configure VAX computers, R1, and provides references to further readings in Tables 1 and 2. The commentary provides a description of the task and a discussion of the rule interpreter with its current states.

Table 1. A DECADE OF SYSTEMS

System	Domain
INTERNIST[42]	diagnosis in medicine
MYCIN[43]	diagnosis in medicine
PUFF[37]	diagnosis in medicine
GUIDON[44]	computer-aided instruction in medicine
VM[45]	measurement interpretation in medicine
SACON[46]	diagnosis in structural engineering
PROSPECTOR[47]	diagnosis in geology
DENDRAL[48]	mass-spectrum analysis in chemistry
SECHS[49]	organic chemistry
SYNCHEM[50]	chemistry
SADD[51]	electronics
EL[52]	circuit analysis
PALLADIO[34]	VLSI design
SOPHIE[53]	computer aided instruction in electronics
MOLGEN[54]	problem solving and planning in genetics
NEWTON[55]	problem solving and planning in mechanics
PECOS[56]	problem solving and planning in programming
DART[57]	diagnosis in computer faults
R1[58]	configuring VAX computers
XSEL[59]	computer sales person assistant
CONCHE[60]	knowledge acquisition
TEIRESIAS[40]	knowledge acquisition for MYCIN systems
HEARSAY-II[61]	speech recognition

Table 2. A DECADE OF TOOLS

System	Domain
AGE[62]	blackboard model tool-kit
EMYCIN[63]	MYCIN tool-kit
EXPERT[64]	diagnosis in medicine
HEARSAY-III[65]	HEARSAY-II tool-kit
KAS[66]	PROSPECTOR tool-kit
META-DENDRAL[67]	DENDRAL tool-kit
NEOMYCIN[68]	computer aided instruction in MYCIN
AMORD[69]	an EL tool-kit
OPS3[70]	an OPS family tool-kit
OPS5[28]	an OPS family tool-kit
ROSIE[71]	a RITA tool-kit

2.3.1 The MYCIN example. MYCIN is an interactive program for medical diagnosis. It produces diagnoses of infectious diseases, particularly blood infections and meningitis infections, and advises the physician on antibiotic therapies for treating those infectious diseases. During the consultation, which is conducted in a stylized form of English, the physician is asked only for patient history and laboratory test results. The physician may ask for an explanation of the diagnosis and the reason a laboratory test result is required.

MYCIN uses a goal-driven rule interpreter to scan a rule memory of about five hundred rules covering meningitis and blood infections. Each rule has a certainty factor that suggests the strength of the association or degree of confidence between the antecedents of the rules and the consequences of the rules. MYCIN has equaled the performance of nationally recognized experts in diagnosing and recommending therapy for meningitis and blood infections. However, it is not clinically used because the human factors of its interface do not save the doctors any time and its knowledge is limited to these two domains.

2.3.2 The R1 example. R1 is a program for configuring VAX computers. It organizes the customer selected components by location relative to other components or to the rooms in which the system will be housed, and it specifies the cabling required to connect pairs of components. This configurer is given only the set of components selected by a customer and

produces the list of missing components, components in each CPU cabinet, components in each UNIBUS box, distribution panels in each cabinet, the floor layout, and cabling requirements.

R1 uses a data-driven rule interpreter to scan a rule memory of about eight hundred rules covering component configuration. These rules are written in the OPS5 programming language and use a direct association between antecedents and consequences. It takes about seventy five seconds on a VAX 11/780 to configure a typical order of about 90 components. R1 has equaled the performance of Digital Equipment Corporation experts in configuring VAX computers and is currently being used in a production mode.

2.4 KBES Summary

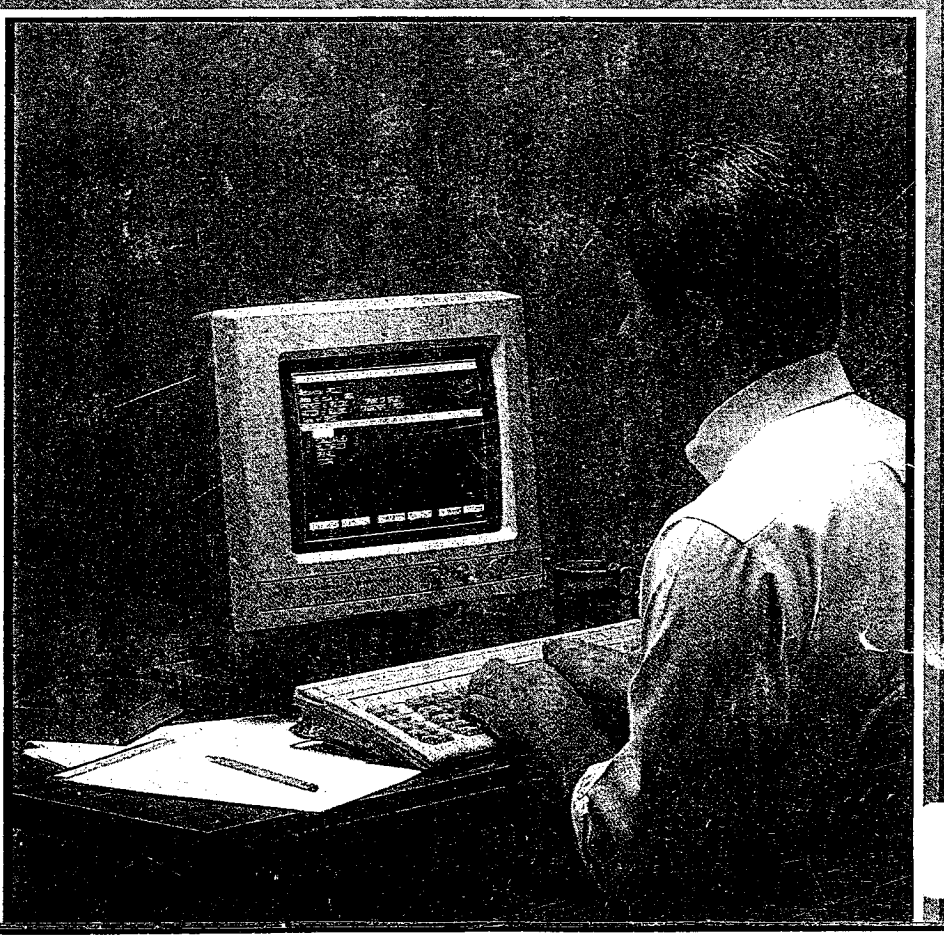
During the past decade many KBESs have been developed by researchers in artificial intelligence that assist experts in solving real-world problems such as interpretation, diagnosis, monitoring, prediction, planning and design. These systems differ from previous efforts in problem solving by effectively coping with the enormous search spaces of alternatives found in real-world problems. The key to success in KBESs is the ability to use domain-rich knowledge to recognize familiar patterns in the current problem state and act appropriately. Development of KBESs is aided by the separation of working memory, rule memory, and rule interpreter, which enables *how* and *why* questions to be asked. Researchers have developed many KBESs whose features have matured and grown into usable engineering tools. Most importantly, this tool is appropriate to the design domain because there are human experts available whose knowledge has been gained through experience and who can teach this knowledge through apprenticeship. Furthermore, the knowledge required to do the task is extensive and requires the type of organization provided by the KBES approach.

2738

EXPERT SYSTEMS

A NON-PROGRAMMER'S GUIDE TO DEVELOPMENT AND APPLICATIONS

PAUL SIEGEL



FIRST EDITION

FIRST PRINTING

Printed in the United States of America

Library of Congress Cataloging in Publication Data

Siegel, Paul.
Expert systems.

Bibliography: p.
Includes index.
1. Expert systems (Computer science)
I. Title.
QA76.76.E95S54 1986 006.3'3 86-6020
ISBN 0-8306-2738-3

is the one for mammal. The top relationship is that *mammal* is *a-kind-of animal*; this relationship acts as a pointer to the animal frame. The skin-cover and activity relations are also shown in the mammal frame. The carnivore and bird frames are related to the semantic network in a similar way.

Two big advantages of frames over semantic networks are that frames may be used for partitioning a complex domain and for storing procedures in addition to descriptive data.

Rules

One of the simplest ways to present knowledge is by rules. Thus, the primary chunk of knowledge given by the mammal frame is:

If animal has hair,
and animal produces milk,
then animal is a mammal.

The rule is presently the most popular method of knowledge representation and is the one I concentrate on for the remainder of this book. The rule is popular because it is:

- ☐ Simple.
- ☐ Modular.
- ☐ Of appropriate size.
- ☐ Procedural as well as descriptive.

Simple. It is easy to express, to understand and to work with.

Modular. Each rule expresses a separate thought and it may be changed or modified without affecting other rules.

Of Appropriate Size. Relations in semantic networks seem to be too detailed. Frames seem to be too broad. Rules are or could be made the correct size. For instance, the data within the bird frame could be part of, not one, but two rules:

1—If animal has feathers,
then animal is bird.
2—If animal flies
and animal lays eggs,
then animal is bird.

Procedural as Well as Descriptive. The rules presented up to now are descriptive. However, rules may refer to procedure as well. This will become obvious as we go along.

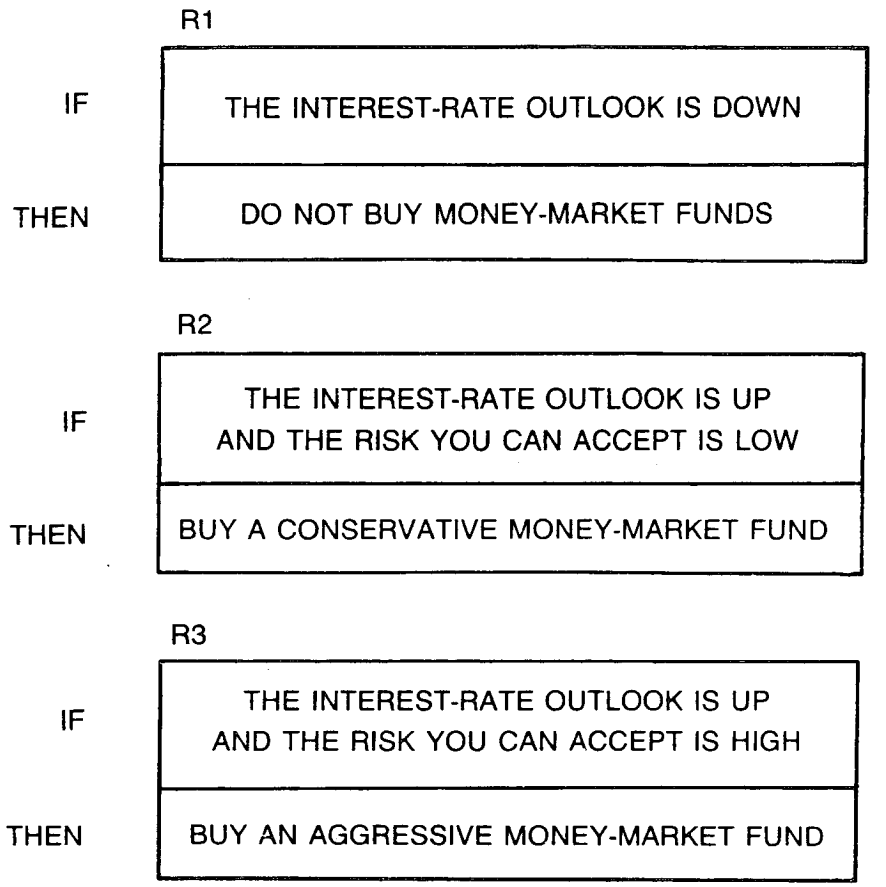


Fig. 1-5. IRA-investment rules.

RULES-OF-THUMB

We often speak of *rules-of-thumb*, meaning guidelines we follow in our reasoning. "An apple a day keeps the doctor away" and "a stitch in time saves nine" are rules-of-thumb.

Rules-of-thumb are easily converted into rules more formally expressed, something which must be done to make the rules manipulatable by computer. Figure 1-5 presents three such rules which are part of an IRA-investment expert system.

A rule consists of two parts: an IF-part and a THEN-part. A rule states a simple relationship: IF certain conditions are true THEN a given conclusion follows. The first rule, R1, states that IF the clause "the interest-rate outlook is down" is true, THEN the conclusion "do not buy money-market funds" follows. Rule R2 is similar except that it has two clauses in the IF-part. It states that IF both clauses, "the interest-rate outlook is up" *and* "the risk you can accept is low" are true, THEN the conclusion "buy a conservative money-market fund," follows. Rule R3 is similar to R2, except that it indicates what should be done if the interest-rate outlook

is up and the risk you can accept is high; in this case, buy an aggressive money-market fund.

Figure 1-6 presents two more rules: R4 and R5. These rules are similar in structure to R1, R2, and R3. Note, however, that the THEN-part of R4 is the same as the second IF-clause of R2, and the THEN-part of R5 is the same as the second IF-clause of R3.

In general, the IF-part may have any number of condition clauses. Similarly, the THEN-part may consist of any number of conclusion clauses. In practice, try to limit the size of both the IF-part and the THEN-part to keep the logic simple.

Rules are modular, pithy chunks of knowledge, which can be replaced or modified without affecting other rules. Rules are the basic building blocks of the *knowledge base* which stores your expertise. Expert systems built around such knowledge bases are flexible, adaptable to changing conditions and able to handle complex problems.

REASONING

Machine reasoning is the path the computer follows as it traces rules through a knowledge base. When the machine sequences forward from facts to final conclusions, or goals, the process is called *forward reasoning* (or forward chaining). When the machine sequences backward from final conclusions, or goals, to facts, the process is called *backward reasoning* (or backward chaining).

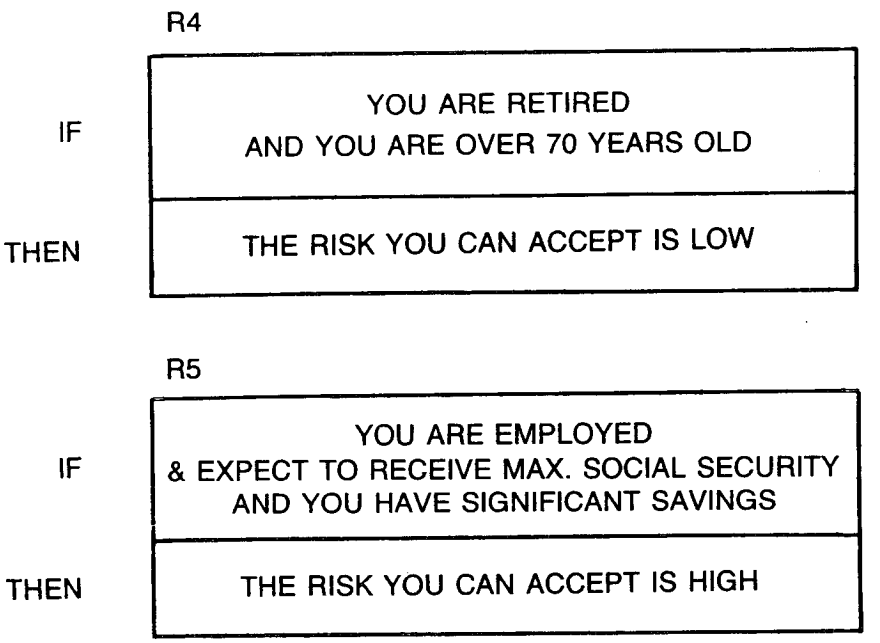


Fig. 1-6. Two more IRA-investment rules.

clusion, it answers with the rule it used. In the previous example, if the client asks how it determined that the client should "Buy aggressive M.M.", it answers by displaying rule R3.

MAJOR CHARACTERISTICS OF EXPERT SYSTEMS

The following are the major characteristics of a rule-based expert system which you can develop with the aid of an Expert-System Builder such as the Personal Consultant without doing any programming:

Knowledge Base

Your knowledge base consists of a collection of rules plus other data representing your knowledge in a specific domain. These rules consist of IF-parts stating the conditions and THEN-parts giving the conclusions.

Separate Reasoning Capability

The reasoning capability is separate from the knowledge base. The reasoning capability, which for the Personal Consultant is primarily backward reasoning, comes built into the Expert-System Builder. But it is copied to a diskette to be made part of the expert system.

System-Client Communication

One way the expert system may activate appropriate rules is by initiating a dialogue with the client. It asks a sequence of questions. Each question is guided by the client's answers to previous questions as well as by the rules and associated information guidelines you, as the developer, have previously stored in the system. Then the expert system communicates advice to the client. The expert system may also communicate directly with other software packages.

Uncertainty

The expert system can handle uncertainty in facts and in rules relating these facts by the use of certainty factors or other functions.

Explanation

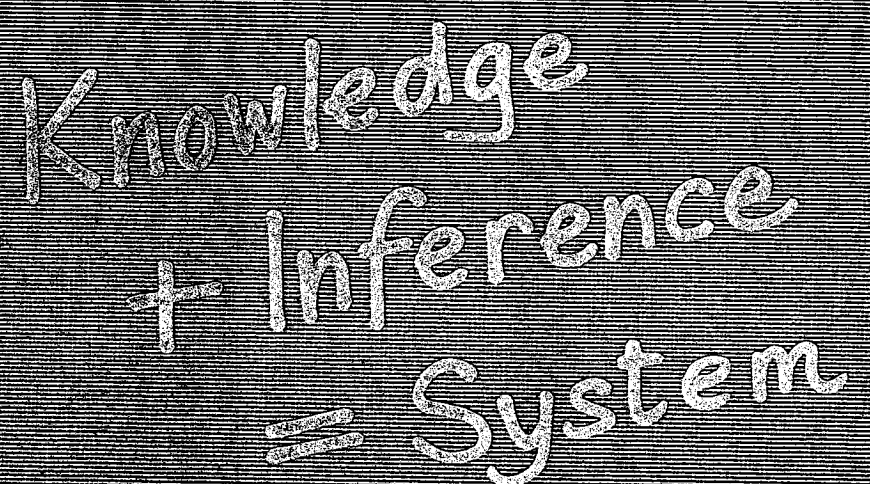
An expert system can explain the reasoning it intends to follow or has followed to reach a conclusion The first is often given in answer to a why, the second in answer to a how.

Expert Systems

Principles and case studies

edited by

RICHARD FORSYTH



CHAPMAN AND HALL COMPUTING

EXPERT SYSTEMS

Principles and case studies

EDITED BY

Richard Forsyth
Polytechnic of North London

LONDON NEW YORK

Chapman and Hall

First published 1984 by
Chapman and Hall, Ltd
11 New Fetter Lane, London EC4P 4EE
Published in the USA by
Chapman and Hall
733 Third Avenue, New York NY 10017

Printed in Great Britain at the University Press, Cambridge

ISBN 0 412 26270 3 (hardback)
ISBN 0 412 26280 0 (paperback)

British Library Cataloguing in Publication Data

Expert systems.
1. Expert systems (Computer science)
I. Forsyth, Richard
001.64 QA76.9.E96

ISBN 0-412-26270-3
ISBN 0-412-26280-0 (pbk.)

Library of Congress Cataloguing in Publication Data

Expert systems.

Bibliography: p.
Includes index.
1. Expert systems (Computer science) I. Forsyth, Richard.
QA76.9.E96E96 1984 001.64 84-15529
ISBN 0-412-26270-3
ISBN 0-412-26280-0 (pbk.)

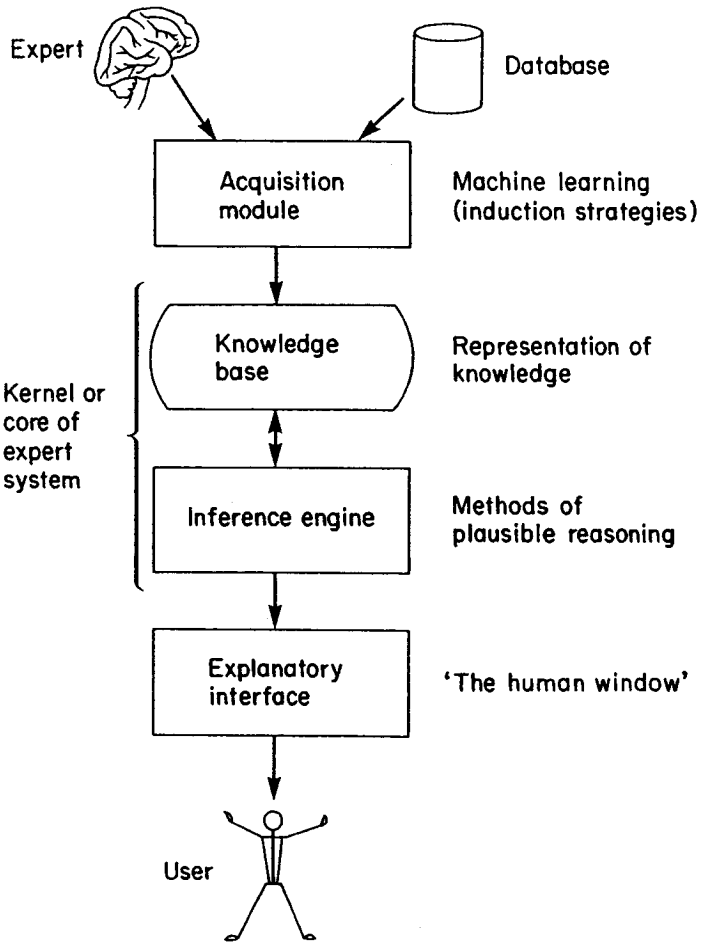


Fig. 2.1 A typical expert system

2.3 THE KNOWLEDGE BASE

A knowledge base contains facts (or assertions) and rules. Facts are short-term information that can change rapidly, e.g. during the course of a consultation. Rules are the longer-term information about how to generate new facts or hypotheses from what is presently known.

How does this approach differ from conventional database methodology? The major difference is that a knowledge base is more creative. Facts in a database are normally passive: they are either there or not there. A knowledge base, on the other hand, actively tries to fill in the missing information.

Production rules are a favourite means of encapsulating rule-of-thumb knowledge. These have a familiar IF–THEN format, for example:

Rule 99

IF the home team lost their last home game, AND the away team won their last home game

THEN the likelihood of a draw is multiplied by 1.075; the likelihood of an away win is multiplied by 0.96.

But remember, these rules are not embedded in program code: they are data for a high-level interpreter, namely the inference engine.

Production rules are not the only way to represent knowledge. Other systems have used decision trees (e.g. ACLS), semantic nets (e.g. PROSPECTOR) and predicate calculus. Since one form of predicate calculus – 'Horn clauses' – is built into PROLOG together with a free theorem prover, this latter representation shows signs of gaining in popularity. At some very deep level all kinds of knowledge representation must be equivalent, but they are not all equally convenient. When in doubt, the best plan is to choose the simplest you can get away with.

2.4 THE INFERENCE ENGINE

There is some controversy in the field between supporters of 'forward chaining' versus 'backward chaining' as overall inference strategies. Broadly speaking, forward chaining involves reasoning from data to hypotheses, while backward chaining attempts to find data to prove, or disprove, a hypothesis. Pure forward chaining leads to unfocused questioning in a dialog-mode system, whereas pure backward chaining tends to be rather relentless in its goal-directed questioning.

Therefore most successful systems use a mixture of both, and Chris Naylor (1983) has recently described a method known as the Rule Value approach which combines some of the merits of both strategies. I call it 'sideways chaining'. (See also Chapter 6 of this volume.)

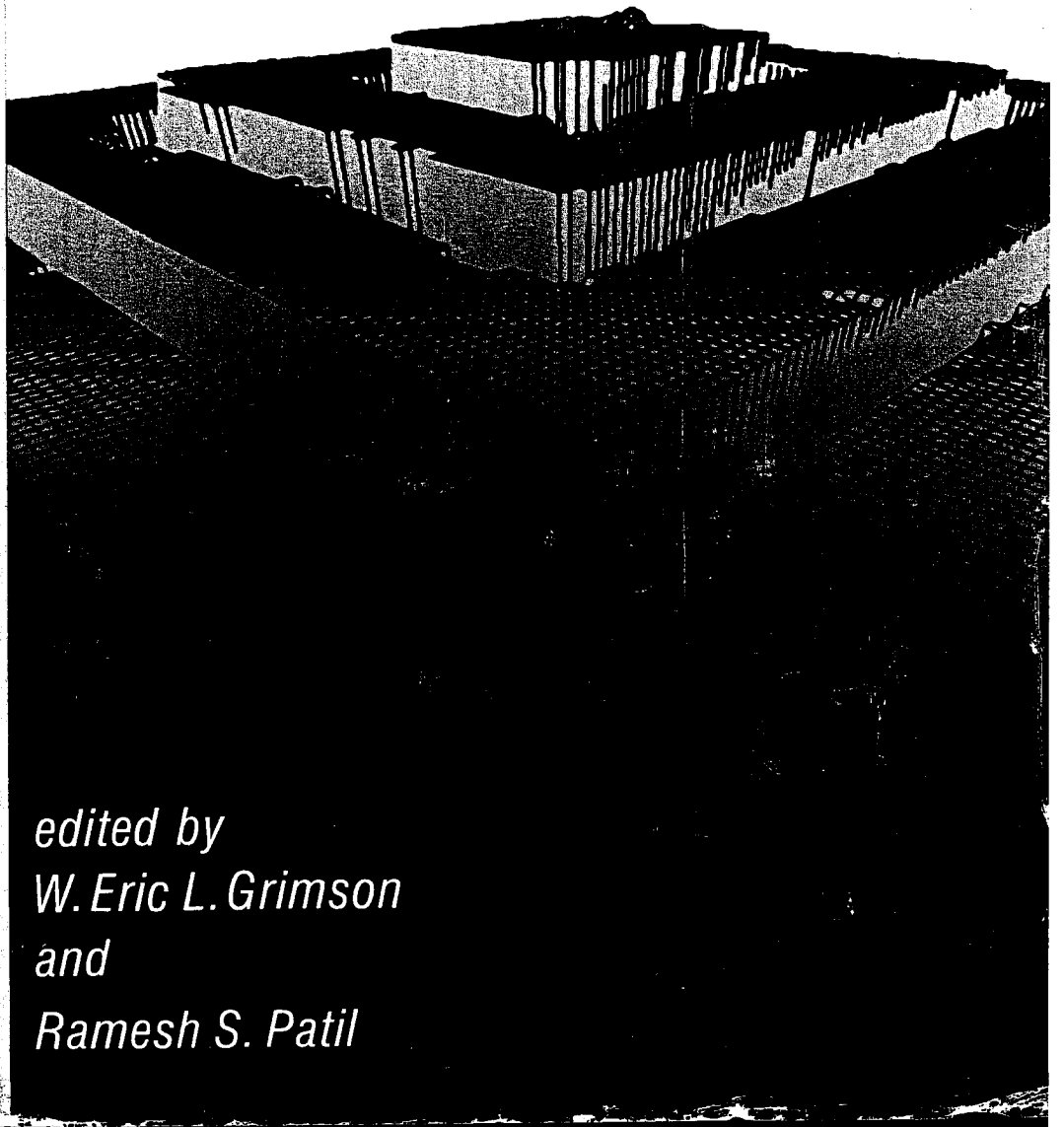
Whether your inferencing procedure works primarily backwards or forwards, it will have to deal with uncertain data and this is where things start to get interesting. For too long computer specialists have tried to force the soft edges of the world we actually inhabit (and understand well enough) into the rigid confines of hard-edged computer storage. It has never been a comfortable fit. Now we have means of dealing with uncertainty, in other words with the real world rather than some idealized abstraction that our data-system forces us to believe in.

Indeed, we have too many ways of dealing with uncertainty! There is fuzzy logic, Bayesian logic, multi-valued logic and certainty factors, to name only four. All sorts of schemes have been tried, and the odd thing is that most of them seem to work.

My explanation for this state of affairs is that the organization of

AI in the 1980s and Beyond

An MIT Survey



edited by
W. Eric L. Grimson
and
Ramesh S. Patil

AI in the 1980s and Beyond

The MIT Press Series in Artificial Intelligence
Edited by Patrick Henry Winston and Michael Brady

Artificial Intelligence: An MIT Perspective, Volume I: Expert Problem Solving, Natural Language Understanding, Intelligent Computer Coaches, Representation and Learning edited by Patrick Henry Winston and Richard Henry Brown, 1979

Artificial Intelligence: An MIT Perspective, Volume II: Understanding Vision, Manipulation, Computer Design, Symbol Manipulation edited by Patrick Henry Winston and Richard Henry Brown, 1979

NETL: A System for Representing and Using Real-World Knowledge by Scott Fahlman, 1979

The Interpretation of Visual Motion by Shimon Ullman, 1979

A Theory of Syntactic Recognition for Natural Language by Mitchell P. Marcus, 1980

Turtle Geometry: The Computer as a Medium for Exploring Mathematics by Harold Abelson and Andrea diSessa, 1981

From Images to Surfaces: A Computational Study of the Human Early Visual System by William Eric Leifur Grimson, 1981

Robot Manipulators: Mathematics, Programming and Control by Richard P. Paul, 1981

Computational Models of Discourse edited by Michael Brady and Robert C. Berwick, 1982

Robot Motion: Planning and Control edited by Michael Brady, John M. Hollerbach, Timothy Johnson, Tomás Lozano-Pérez, and Matthew T. Mason, 1982

In-Depth Understanding: A Computer Model of Integrated Processing for Narrative Comprehension by Michael G. Dyer, 1983

Robotics Research: The First International Symposium edited by Michael Brady and Richard Paul, 1984

Robotics Research: The Second International Symposium edited by Hideo Hanafusa and Hirochika Inoue, 1985

Robot Hands and the Mechanics of Manipulation by Matthew T. Mason and J. Kenneth Salisbury, Jr., 1985

The Acquisition of Syntactic Knowledge by Robert C. Berwick, 1985

The Connection Machine by W. Daniel Hillis, 1985

Legged Robots that Balance by Marc H. Raibert, 1986

Robotics Research: The Third International Symposium edited by O. D. Faugeras and Georges Giralt, 1986

Machine Interpretation of Line Drawings by Kokichi Sugihara, 1986

ACTORS: A Model of Concurrent Computation in Distributed Systems by Gul A. Agha, 1986

Knowledge-Based Tutoring: The GUIDON Program by William Clancey, 1987

Visual Reconstruction by Andrew Blake and Andrew Zisserman, 1987

AI in the 1980s and Beyond: An MIT Survey edited by W. Eric L. Grimson and Ramesh S. Patil, 1987

AI in the 1980s and Beyond

An MIT Survey

Edited by W. Eric L. Grimson
and Ramesh S. Patil

The MIT Press
Cambridge, Massachusetts
London, England

PUBLISHER'S NOTE

This format is intended to reduce the cost of publishing certain works in book form and to shorten the gap between editorial preparation and final publication. Detailed editing and composition have been avoided by photographing the text of this book directly from the author's prepared copy.

This book was printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

AI in the 1980s and beyond.

(The MIT Press series in artificial intelligence)
Includes index.
1. Artificial intelligence. I. Grimson, William Eric Leifur. II. Patil, Ramesh S. III. Massachusetts Institute of Technology. IV. Series.
Q335.A42 1987 006.3 87-3241
ISBN 0-262-07106-1

Knowledge-Based Systems: The View in 1986

Randall Davis

Three things of interest stand out in reviewing the state of the art of knowledge-based systems in 1986. First, two technologies for building these systems — rules and frames — have become reasonably familiar and widely applied. This paper reviews them both briefly, exploring what they are and why they work, both as a way of characterizing what we currently know well and as lead-in to a discussion of some of their problems and limitations.

Second, we are seeing a strong surge of commercialization of the technology, leading to some interesting developments in the system-building tools that are being created, and an interesting evolution in the role of the technology within organizations.

Third, new ideas have begun to emerge about how to build these systems, ideas that are being referred to as model-based reasoning, or "reasoning from first principles." To illustrate them, I'll describe some of the work that's going on here in my group at MIT.

Knowledge-Based Systems

Any time a new technology appears, new jargon emerges to describe it. What are we to make of the term "knowledge-based system"? The term is really an answer to the question "What makes an expert an expert?" That is, why is someone who is good at what they do *that* good at it? Do they think differently than the rest of us? Do they think faster than the rest of

us? Is there something inherently different about an expert's thought and problem solving?

Calling something a knowledge-based system is in part a commitment to believing that that's not true. It is instead a commitment to believing that the primary source of expertise is knowledge, that problem solving skill arises primarily from knowledge about the task at hand. That knowledge includes both facts about the problem and a wide array of problem solving strategies that experts accumulate over time.[1] Programs built in this inspiration are knowledge-based in just this sense: their power arises not from their speed as computer programs, nor their ability to retain almost endless detail; it is instead based on their sizable store of knowledge about the task domain.

Rule-Based Systems

Figure 1 shows an extract taken from a pamphlet on tax planning, specifically selecting an organization structure to maximize the benefit from regulations concerning tax-loss pass-through. This extract, perhaps unwittingly, takes this very same view. Consider the first paragraph and note in particular the first three words: *Skilled practitioners know...*, reflecting the implicit view that people are good at a task because they know something about it.

> Skilled practitioners know how to qualify a business for pass-through treatment while avoiding potentially unpleasant consequences.
>
> There are two classes of partnerships: general and limited. General partnerships have the advantage of being much easier to draw up and of having no problems qualifying as partnerships for tax purposes. But they also have the significant disadvantage of establishing unlimited liability of the partners for the debts of the partnership.
>
> If the partners do not care about liability, or liability insurance is adequate, then a general partnership is easiest. If liability insurance is not adequate and the investors do not care about management participation, then a limited partnership is very likely the right idea.

Figure 1. Extract from a pamphlet on tax planning.

[1] Other factors affect performance as well, of course: things like practice, for example, allow the expert to perform smoothly and quickly. The claim here is that the primary, but not sole, foundation for problem solving skill is knowledge of the task.

Consider the second paragraph, which tells us facts about the world. In particular it tells us there are two classes of partnerships, general and limited, and then goes on to tell us some additional facts about each of those.

Finally, the last paragraph gives us some rules for thinking about and solving problems in this domain; in particular choosing one or another of these organizational structures to to take advantage of tax laws on handling business losses.

This text rather nicely reflects much of the mindset of knowledge-based systems: it notes the dependence of skill on knowledge and then illustrates two forms of that knowledge: facts about the domain and rules for solving problems in the domain.

We are in general accustomed to the idea that computers can be used to help do arithmetic: it's relatively easy to give a machine a formula for compound interest or use it to figure out budgets. We are somewhat less accustomed to the notion that we can take information like that in Figure 1, put *it* inside such a machine and have the machine use it in solving problems.

Part of the contribution of the current knowledge-based system technology is a way of doing this that is relatively easy to use. We can, for example, take the last sentence in Figure 1 and reformat it just a bit to make its content more explicit (Figure 2). As Figure 2 shows, the rule indicates that if three conditions are met, then there is some evidence that a limited partnership is the right choice.

IF

you are considering a partnership, and
liability insurance is not adequate, and
the investors do not care about management participation,

THEN

there is strong evidence (.8) that a limited partnership
is the right structure.

Figure 2. Simple if then-decision rule.

The important point here is that we have a way of capturing *symbolic inference rules*, rules of inference that can capture the thinking and problem

solving of experts. While mathematical models and calculations can be quite powerful, much of what we know about the world is not well captured by numbers, much of our reasoning is not well modeled by arithmetic. Rules of the form shown in Figure 2 can be of considerable utility for capturing and putting to use this other form of knowledge. They allow us to capture some of the ways people think about a problem and embody those ideas inside the machine.

Systems built in this manner are called *rule-based systems*; their standard architecture is illustrated in Figure 3. The two most fundamental components are the inference engine and the knowledge base. The knowledge base typically contains a few hundred to a few thousand rules of the very simple if-then decision form shown above. The inference engine is a mechanism that uses the knowledge in the knowledge base, applying it to the problem at hand.

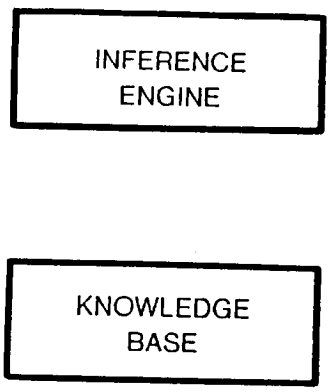


Figure 3. Architecture of a knowledge-based system.

Perhaps the most common way of using the knowledge is *backward chaining*.[2] Suppose we were trying to use rules of the sort shown in Figure 2 to determine what structure to select for an organization. The system will retrieve all the rules that are relevant to that conclusion, namely all the rules that tell us something about that particular topic. Rule 27 (Figure 2) is one such rule because it concludes about an organizational structure.

[2] For reasons of space the description here of this process is much abbreviated. For a more complete description see [Davis77].

We can make the conclusion shown in that rule if each of the three clauses in its "if" (or "premise") part are true. How do we determine the truth of each of these? We start all over again doing exactly the same thing, namely retrieving all the rules that are relevant to the first clause (about liability insurance) and trying each of those rules in turn (Figure 4b).

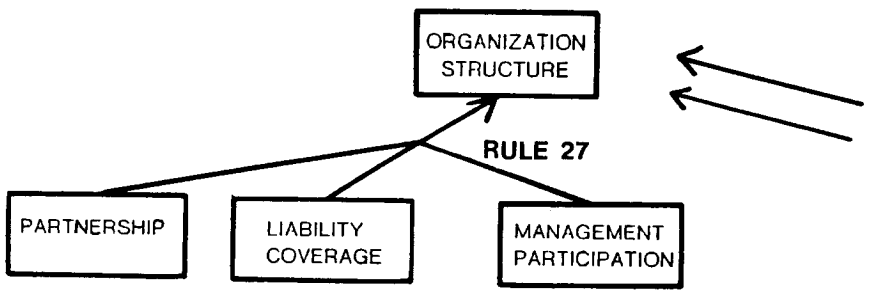


Figure 4. Backward chaining, step 1.

This is called backward chaining because we're starting with what we're after (organization structure) and retrieving rules relevant to that topic; this in turn brings up new topics we have to deal with. We then retrieve rules relevant to those new topics, working backwards toward ever more basic pieces of information. Eventually, of course, the systems encounters topics for which there are no rules. At that point it requests specific facts from the user in order to determine the truth or falsity of the premise clause under consideration.

Eventually the system will determine the truth of all three clauses of the rule; if all have been satisfied then this rule will fire and make one conclusion about organizational structure. Since rule 27 offers evidence that is strong but not definitive, the system will then go on to try all the other rules relevant to that topic, in effect collecting all the "opinions" it can about what organization to choose.[3]

From the user's point of view, interaction with the system looks like the trace shown in Figure 5. The system is working its way backward through the rules, encountering topics for which it has no rules, asking questions of the user via the dialogue shown here.

[3] See [Shortliffe75] for one description of how to deal with the uncertainty captured in phrases like *strong evidence* and *weak evidence*.

1) What is the expected startup income (loss) of the business in the first fiscal year?
** (182000)

2) What is the investor's marginal tax bracket?
** 50

3) Assuming losses can be passed through, will the losses be useful to the investor?
** Y

4) What is the business manager's expected marginal tax bracket?
** 45

5) Is the business liability coverage adequate?
** UNKNOWN

Figure 5. Sample trace of the system in operation.

There are two particularly interesting characteristics of these systems. First, they work. It was not obvious when efforts began in this area more than a decade ago that one could in fact capture non-trivial bodies of expertise by collecting somewhere between a few hundred and a few thousand very simple rules of the form shown. It's an empirical result, and a somewhat surprising one, that it simply turns out to work and in fact works for a fairly wide variety of applications.

The second interesting characteristic of these systems is shown in Figure 6. Here, in the middle of another consultation, instead of answering the question the user has asked "why," that is, "Why are you asking me this? Why does it make sense to request that piece of information?"

The system's answer is shown, saying in effect, "Here's what we're after (organization structure), we've already determined that the first two items are true, so if we also know this third item, then we'll be able to make the conclusion shown."

There are two interesting things going on here. First, we have in effect stopped the program in the middle of what it was doing, said "What are you doing?" and the answer that came back was comprehensible. It's comprehensible in part because, in the most general sense, what the program is doing is symbolic inference, not arithmetic; problem solving, not calculation.

7) Might the investor want to manage the enterprise at some point?
** WHY

We are trying to determine the right structure for the organization.

It has already been determined that
[1] you are considering a partnership, and
[2] liability insurance is not adequate

Hence if it is also true that
[3] the investors do not care about management participation,

then we can conclude that there is strong evidence that a limited partnership is the right structure.
[RULE27]

Figure 6. Answering a "why" question.

Imagine by contrast stopping a program that does linear programming or Bayesian decision theory or some other variety of fundamentally mathematical operation and saying "What are you doing?" What could it possibly say?

Mathematical methods, as powerful as they are, and as appropriate as they are for a number of problems, simply are not and never were intended to be models of how people think. One of the advantages of a knowledge-based system is that it is so structured, that it does allow us to capture the way people reason.

The second interesting point is *transparency*. The program is not a "black box." We can in effect look inside it and see how it works, see what it's doing, examine the line of reasoning being used.

That in turn is important because it leads to a notion of *accountability*. The system does not simply supply an answer, but can in addition tell us something about why that answer seemed appropriate. Hence in addition to getting a result, we can also get an accounting of how that result arose. As a result, we need not accept the answer on a take it or leave it basis; we can instead examine it in the context of its explanation and come to understand both the outcome and its rationale.

Figure 7 shows another form of explanation. At the end of a consultation the system will display its answer about how it thinks the organization ought to be structured. We might then discover that the answer we would

have expected to see in fact doesn't appear. It is possible at that point to ask a question of the sort shown in Figure 7. We are in effect asking the system why it didn't do something else.

** WHY DIDN'T YOU SUGGEST AN S CORPORATION?

There are three rules that indicate that the structure of the organization should be an S corporation (rules 98, 43, 17).

However, all of them require that the number of shareholders is no more than 35. You said (in answer to question 24) that the number of shareholders is 43.

Figure 7. Answering "why not".

It responds by telling us which rules could have come to that answer, then explaining why none of them were applicable. Several things are worth noting here. First, in addition to *using* its knowledge to solve the problem at hand, the system is also capable of *examining* that knowledge. The system isn't applying the rules here (as it did to generate its original answer), but is instead examining the knowledge base it to determine how it might have arrived at the alternative answer.

Second, this very simple form of introspection gives us a very interesting consequence: the system can not only tell us what it did, it can tell us what it *didn't do and why.* Consider the consequences of that in the context of due diligence.

Third, this is another illustration of the notion of accountability: the system is not only indicating what it did, but is explaining as well why other alternatives were not appropriate.

Finally, consider in the long term the impact of systems like this in decision support: they not only facilitate the decision making process, but can as well facilitate the communication of the reasoning behind the decisions.

Most of the practical application systems built to date have been constructed using rules, in part because this is the simplest and best understood technology, though it is by no means free of shortcomings. We consider limitations of the technology below.

Frame-Based Systems

The second fairly well-established approach is based on the idea of *frames*

as a way of expressing knowledge. Since they were first suggested in rather high level terms some years ago [Minsky75], frames have come to mean a number of different things and have been used in several different ways. Despite the diversity, four basic concepts appear common to the different conceptions:

- A frame contains information about a prototypical instance.

  In medical applications, for example, a frame is used to describe a prototypical or "classic" case of a disease.

- Reasoning with frames is a process of matching prototypes against specific individuals.

  For example, a specific patient is matched against a collection of disease prototypes to find the closest match.

- Frames are often organized into taxonomic hierarchies, providing an economy of knowledge representation and reasoning mechanism.

  A small segment of a diseases hierarchy is shown in Figure 8; we consider below how this can be used.

- The information within a single frame is typically expressed as a collection of *slots*, each of which has one or more *values* in it.

  This aspect of frames offers a convenient way of expressing many of the simple facts about a domain.

The knowledge base for a system like this typically consists of a large number of frames, each capturing a single prototypical description. In a medical system, for example, a frame would describe one disease, such as a classic case of viral hepatitis.[4] Part of the task of building the knowledge base is to determine what the prototypes are (e.g., what all the diseases are), and then how best to characterize each (e.g., what the presenting symptoms or other characteristic features are). This is typically a large and difficult undertaking.

The basic task is then to match the set of frames against the collection of facts about a specific patient, to determine which frame or frames best matches. In more general terms, we need to compare the prototypical cases against a specific instance to determine the closest match.

The matching process proceeds in two phases: generating hypotheses by reasoning from observed symptoms to possible diseases, then evaluating

[4] As in the case of rule-based systems, the description given here of frame-based reasoning and representation is necessarily abbreviated. For a description of a real frame-based system, see [Pople82]; for more on the uses of frames for building expert systems, see [Fikes85].

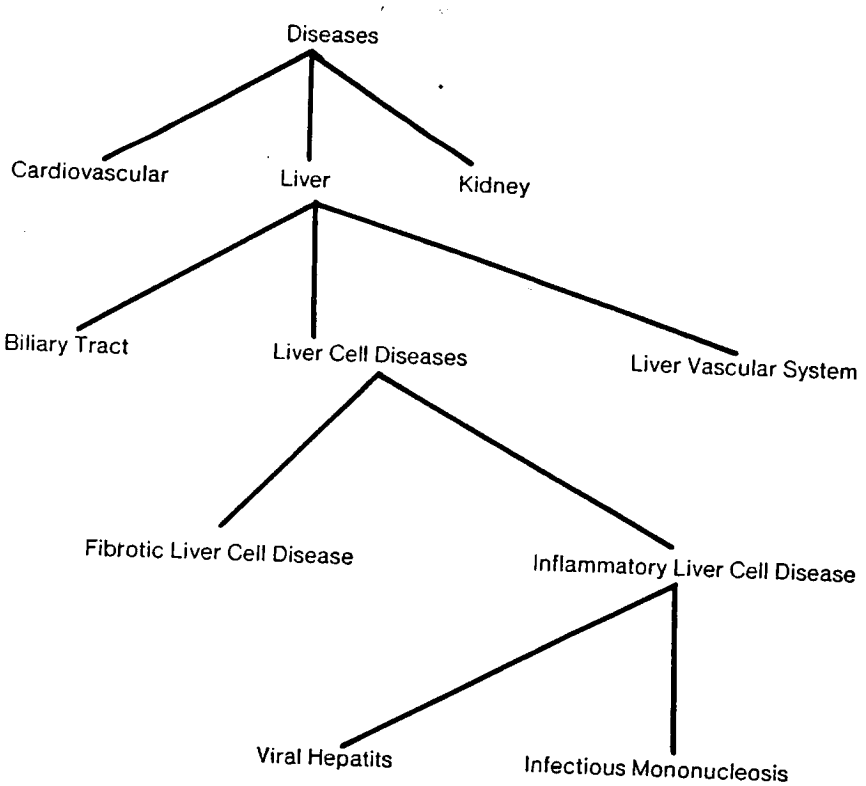


Figure 8. Segment of a disease hierarchy.

hypotheses by determining how well each accounts for the observed symptoms. In the hypothesis generation stage we have a collection of symptoms and need to ask what diseases they suggest. The basic question is, how well do the observed symptoms *support* any particular hypothesis?

One illustrative (but very simple) way to answer this is simply to count symptoms: if the patient has nine of the ten classic symptoms of viral hepatitis, we can be fairly confident that it is a plausible hypothesis. If, on the other hand, the patient only has one of the ten symptoms, we're much less likely to entertain that hypothesis.

The second stage goes the other way around. The question now is, given the diseases hypothesized, how well does each *account for* the observed symptoms? That is, suppose we believe for the moment that the patient actually has viral hepatitis. What consequences does this have for the set of observed symptoms? Does that account for all the symptoms, or

is there "undershoot", i.e., are there still some "left over"? If so, then we should start the process over again, this time matching the frames against only the "left over" symptoms, to see what else may be present.

There may also be "overshoot": are there classic symptoms of viral hepatitis that ought to be present and that don't show up in the current patient? If so, then the hypothesis overpredicts and we need to decide how to respond. One possibility is to rule out viral hepatitis: if the predicted symptom is a necessary finding in anyone with the disease, then its absence rules out the possibility of the disease.

Organizing frames into a taxonomy can add two things to this process. First, the concept of *inheritance* offers an economy of representational mechanism. It is based on the principle that many things true of a general category are true of the more specific subcategories as well. In Figure 8 for instance, many of the things true of liver cell disease are true for all subcategories of that disease as well. The basic insight is to store those things only once, as "high up" in the taxonomy as possible. We need not repeat in our description of viral hepatitis all those things also true of its supercategory, inflammatory liver cell disease. Viral hepatitis will "inherit" those from its supercategory.

This is relatively easy to accomplish in simple cases, but the general case is fairly difficult: there are numerous subtleties to inheritance, such as handling exceptions, that are not yet well understood.[5]

A taxonomy also offers the opportunity to reason by successive refinement: the system can begin by using frames near the top of the hierarchy, in effect classifying into a broad category, then refine this by using frames found at the next level beneath that category. This permits a focusing of effort that can be very important in a large knowledge base.

This is a much simplified version of how such programs work, but it gives a general sense of their operation. Real systems typically have considerably more sophisticated ways of generating and evaluating hypotheses, dealing with interactions between hypotheses, etc.

Roles, Successes and Limitations

Names can at times be problematic and that has been the case when these systems are referred to as *expert systems*. The term conjures up a wide variety of expectations about performance and role, for example. It is im-

[5] For more detail on this, see [Etherington83].

portant to note therefore that they need not function as experts in order to be very useful. There is in fact a whole spectrum of roles that are relevant: these systems can and have been built to function as both assistants and colleagues.

For any system in the foreseeable future, we will have a combination of human and machine. The real issue is, where on the line of collaboration (Figure 9) are we going to draw the separation of responsibility? If the line appears nearer the "machine" end, with the human responsible for most of the problem-solving, the system will be functioning as an assistant. If the line appears more toward the middle, it becomes a collegial relationship with both contributing equally. If the line can be pushed nearer to the human end, with the machine taking on most of the task, then we have something that may truly deserve the term expert system.

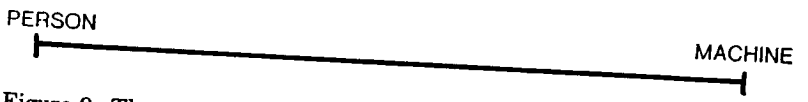


Figure 9. The spectrum of roles for a knowledge-based system.

The important point is that there are extensive pragmatic benefits at every point along that spectrum. The system need not be as good as an expert to be useful; even an intelligent assistant can be of great utility. Such a system can serve to offload routine cases, freeing up a human expert's time to focus on the more difficult and challenging problems. It in effect becomes an "intelligence amplifier", allowing us to use the scarce resource of the expert's time in a much more effective manner.

It's also important to recognize that there are a number of well known limitations in this technology. An extensive list appears in Figure 10; a few will serve to illustrate. There is, for example, the problem of scope: we have to define these problems narrowly enough to make the knowledge base construction task feasible, yet ensure that we don't define away the difficult part of the problem. If the task is *too* narrowly defined, all of the problem solving is done in simply knowing enough to come to this particular source of expertise.

There are also substantial difficulties surrounding the problem of inexact reasoning, problems recognized in mathematics and psychology some

Scope

How much of the problem is solved in coming to the expert?

Getting Religious

Some information is not well captured by rules of frames.

Simple Representation

The standard attribute/object/value representation is very weak.

Multiple Diseases

Most systems assume no interaction between them.

Inexact Knowledge

A difficult problem, both definition and execution.

Common Sense

Pervasive in human reasoning, yet elusive.

Accumulating Knowledge By Cases

Is there a more fundamental understanding than individual rules for individual situations.

Character Of The Knowledge

Reasoning qualitatively and reasoning causally are both important in human reasoning, but difficult to capture.

Learning

Wouldn't it be better to just have the system discover what it needed to know.

Figure 10. Limitations of the current technology.

time before knowledge-based systems arrived. One fundamental problem is to specify exactly what is meant when a rule says "probably", "maybe", or "very likely." Does this refer to probability or to strength of belief? That is, does the rule indicate how likely it is that something is true, or does it refer to how strongly the rule author believed it? The two are quite different, as any gambler knows.

A second difficulty lies in defining the "arithmetic" to use in combining them. What does "maybe + probably" add up to? Very likely? Very likely. The problem has been attacked by mathematicians and philosophers; one indication of its difficulty is the large number of solutions that have been suggested.

As one final example, there is the difficult problem of learning. Systems today are typically built by debriefing experts, in effect by asking them to tell us what they know about the task at hand. This is difficult, time consuming, and not incidentally, assumes that there exists a willing expert.

One clearly desirable alternative is automating induction: we would like to be able to take a large collection of case studies — example problems with answers — give them to the program, and have *it* figure out what it ought to know. It's a fine aspiration, but a difficult problem, with much active and exciting research going on right now. But the message here is that this is still a long term dream: current practice on any non-trivial problem is still a process of finding a willing expert and spending a long time talking.

Commercialization of the Technology

A second major issue worth note in reviewing the current state of the art in knowledge-based systems is the fast-growing commercialization of the technology. One of the most interesting trends in this area is an evolution in the perception of the technology from something used strictly in support of existing manufacturing businesses, to one that is providing the basis for new products and new businesses.

The earliest applications of knowledge-based systems have been as in-house production aids. Systems like XCON, ACE (built by AT&T for cable maintenance), and others are designed to help the corporation carry out its existing function, perhaps faster and/or cheaper, but still in pursuit of its traditional agenda.

More recently we are beginning to see the development of systems designed not for internal consumption, but for sale as finished products, at times allowing an organization to move into a new line of business. An important characteristic of these systems is that AI is the supporting technology, not the product. Unlike the general purpose tools (expert system "shells" and other toolkits) currently appearing in the market in large numbers, they are designed to meet specific applications needs that AI can support, rather than simply to offer the technology and leave the application to the user.

One example, interesting because it characterizes a growing class of such systems, is a program called **PLANPOWER**^{TM}, built by Applied Expert Systems (Cambridge, MA) to do individual financial planning. More generally, the task is matching consumer product needs against an exploding marketplace of choices.

The need is particularly acute in financial services. New products appear with distressing frequency (zero-coupon bonds, index funds, sector funds, etc.). As a result of deregulation new players are entering the game, with banks selling life insurance, brokers selling CD's and insurance salesman selling tax shelters. The profusion of new products is overloading the traditional financial advisor, while the new players are hard-pressed to find enough trained personnel to provide the desired level of service.

PLANPOWER embodies the financial planning skill of expert human planners, discovered through extensive debriefing, aimed at finding out how to match needs against available products, and how to plot a course from where the client is at the moment in their financial life to where they would like to be in the future. The system functions in the traditional decision support system role, not replacing, but leveraging the time of financial planning professionals. Time saved in backoffice plan preparation translates directly into more time spent with clients and more time available to spend with new clients.

The application is also interesting because it involves combining both traditional arithmetic calculations (e.g., cashflow, tax projections, etc.), with the judgemental knowledge characteristic of knowledge-based systems (e.g., knowing what kinds of financial planning strategies to try).

Most important, however, are several interesting characteristics in the design and conception of the system. It is designed for use by service professionals, not by engineers, manufacturing or technical staff. It is designed for sale to and use by outside organizations, not for internal use and con-

sumption by the organization that created it. And it permits organizations to enter new markets by offering a way to distribute expertise to a large field force. For the traditional players in the market, it offers a way to deal with the information explosion that overloads their existing service providers. For the new players, it offers a means for building a large and highly skilled service network quickly and relatively inexpensively.

There are other, somewhat more speculative consequences that we may see in the long run. It may prove possible to use such systems to distribute corporate policy. Suppose an organization decides to make a significant change in its approach to a problem or a market. It can send out memos, the usual 10-100 page documents, hoping that someone reads them. In time, it may instead be able to send out new floppy disks, reloading and updating the knowledge base in the field, in one quick and effective step changing the way the entire field staff approaches the problem.

This is also one example of information technology used as a strategic weapon. Given, for example, the typically extensive databases compiled by financial planners about their clients, new product marketing strategies become possible, including automatic, very narrow targeting of new products to a carefully selected subset of the customer base.

What are we likely to see in the near future? More systems like **PLANPOWER** will appear, that is, more systems designed for non-technical, service-oriented end-users, systems that permit companies to attack new markets and provide new services.

The system building tools are going to get better. As useful as they are currently, it still takes a non-trivial amount of skill to use them well, in part because the art of system building is still new enough that we don't know how to make it a routine operation accessible to anyone.

We're going to see many "mundane" applications. Despite the image of AI as high technology, it may be that the most commercially significant applications are hardly the stuff of science fiction. Campbell's Soup, for example, was faced recently with the imminent retirement of someone with 44 years of experience in running one of its soup production lines. Rather than lose that body of skill, they worked with the expert to capture what he knew about that specific task and embodied it in a rule-based system. Running a soup cooker well is far from dazzling high technology, but it does save significant time and money. I think we're going to see many more of these apparently mundane but in fact quite important applications.

Finally, and, I hope, quite soon, we will see a much more subdued

acceptance of the technology. AI and knowledge-based systems are exciting, fundamentally new tools, but in the long term they need to be added to the existing kit of techniques for applying information processing. Much as with personal computers, spreadsheets, and other technologies, they have both their important roles and their limits. The sooner they are accepted as such the better.

Reasoning "From First Principles"

The final major issue I'd like address is the technology that we believe will provide the foundation for the next generation of knowledge-based systems. I'll do this by examining some of the research currently going on in my group at MIT.

The fundamental issue is solving problems in a style called reasoning "from first principles." Perhaps the easiest way to understand this is to contrast it with traditional knowledge-based systems, paying particular attention to the character of the knowledge captured in each.

Traditional systems are built by identifying and accumulating rules of the sort discussed earlier, rules that can be characterized as *empirical associations*. They are connections, typically between symptoms and diseases, which are "true" in the sense that they have been observed to hold empirically, but typically can't be explained in any more detail.

As one example, consider the following rule that we might find in a medical diagnosis system.

If the age of the patient is between 17 and 22
the presenting complaint is fatigue,
the patient has a low-grade fever,
then the disease is likely (.8) to be mononucleosis.

In some sense this rule is "true"; doctors have noticed the association often enough to know that it is is worth taking note of. But if we ask *why* it is true, we often find that the answer is something like, "The ultimate etiology is yet to be determined," (medical jargon for "Damned if we know").

The important points here are the associational character of the knowledge, its lack of more fundamental explanation, and it typical origin: the accumulated experience of practitioners. Such rules can be an effective way to capture expertise in domains where there does not yet exist a more fundamental understanding of cause and effect.

Suppose now that we tried to used this rule-based approach to do computer diagnosis in the literal sense: i.e., hardware maintenance, determining what's wrong with my computer when it breaks.

A number of difficulties would appear. First, the set of rules would be strongly machine-dependent. The particular set of foibles and symptoms common to one model of computer are often quite different from those of another, meaning the work of accumulating such rules must begin again for each machine. That in turn presents a significant problem, because the design cycle for new machines is growing short enough that there may not be time to accumulate all the needed rules. Field service staff will as a result constantly be catching up, struggling to stay ahead of the introduction of new machines as their recently acquired rules soon become obsolete.

The same problem crops up in a less dramatic way in the design upgrades made to machines already in use. The changes to the design may be quite small, but this can lead to significant changes in the surface symptoms, the way in which problems manifest themselves. Once again, the problem is rapid obsolescence of the knowledge.

The traditional approach has been reasonably successful in medicine because the model line is quite limited (only two basic models) and it has been relatively stable over the years (the design cycle is rather longer). As a result we have had the time to accumulate the relevant knowledge base, one that remains reasonably stable.

Finally, there is the problem of novel bugs. Traditional rule-based systems accumulate experience, embodying a summary of all the cases the expert has seen. If a problem presents unfamiliar surface manifestations, by definition the rule-based approach will be unable to deal with it, even if the underlying cause is within the scope of the system's competence (i.e., the system can identify it given other, more familiar symptoms).

Troubleshooting

For all those reasons the traditional rule-based technology was found to be insufficient to attack the problem of troubleshooting electronic hardware. Instead we sought to reason from a description of structure and behavior, in the belief that there were fundamental principles that could be captured. Such a system would have a degree of machine independence, would be quickly deployable, given the relevant schematics, and could reason from any set of symptoms, familiar or novel.

The specific problem of troubleshooting is part of a larger research strategy (Figure 11). Sitting at the center are languages for describing structure and behavior. Around the outside is a whole range of specific problems we are working on, including troubleshooting, test generation, test programming, and design debugging.

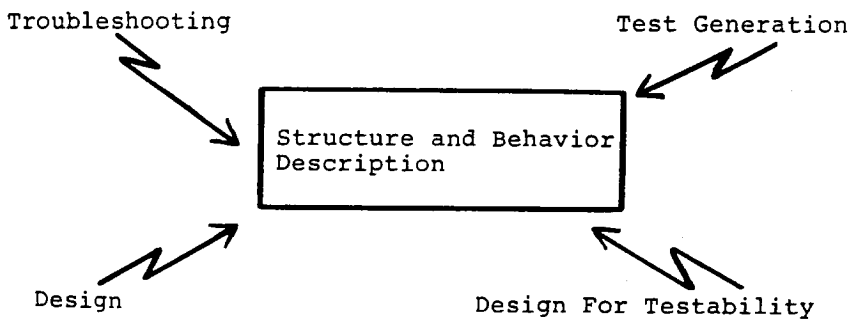


Figure 11. Overall research strategy.

The best way to understand our approach is to consider one specific example. The description that follows is a much abbreviated version of an example that appears in complete detail in a special issue of *Artificial Intelligence* [Davis84].

Consider the electronic circuit pictured in Figure 12, made up of three multipliers and two adders. It doesn't do anything particularly useful except provide a simple example to explain some of the interesting and difficult aspects of this problem.

Given the inputs shown, straightforward simulation will indicate that both outputs should be 12. Now suppose we also had a physical model of the device, gave it the same inputs, and measured what came out. Suppose that at output G we find a 12 as expected, but get a 10 at output F. Let's focus on this discrepancy between what we expected (from simulating the correct behavior) and what actually appeared, attempting to discover which component(s) might be broken in such a way as to account for the symptoms.

One way of attacking this problem by reasoning from structure and behavior starts by asking an obvious question: Why is it that we expected a 12 at F? There are three reasons: (a) we expected the box labeled ADD-1 to act like an adder, (b) we expected its X input to be 6, and (c) we

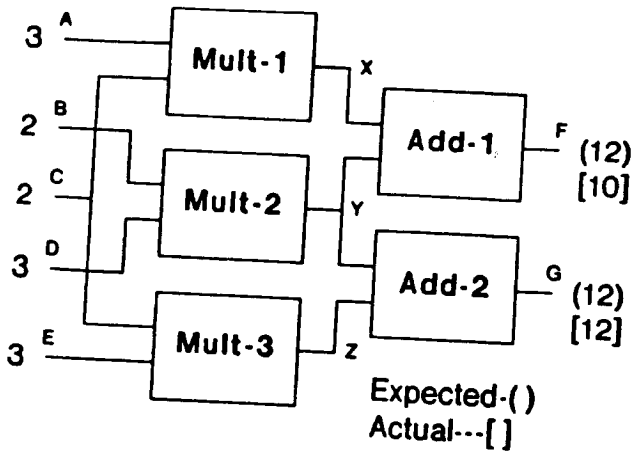


Figure 12. A simple troubleshooting example.

expected its Y input to be 6. If all those three things had been true, the value at F would have been 12. So one of them must be false. (To be more precise, *at least* one of them must be false, but we'll deal with just one for the moment.)

Consider each of those assumptions in turn. Suppose only (a) is false: in that case both X and Y are 6 (by assumption), but ADD-1 is not functioning as an adder. We don't know what it *is* doing, all we know are the symptoms (both inputs are 6, output is 10). Hence one possibility that is globally consistent with all the symptoms is that ADD-1 is broken in the fashion noted.

Now suppose only (b) is false. In that case, by assumption, ADD-1 is functioning properly and its Y input is 6, and we measured the value at F to be 10. Under those circumstances we can infer that the X input of ADD-1 must have been 4. That's interesting, because now we have another discrepancy, this time between the 6 predicted by simulation as the result of MULT-1, and the 4 inferred from the output observed at F. We can now ask the same question all over again ("why did we expect a 6 at X?"), and

discover that the problem may be a malfunction in MULT-1 (inputs 3 and 2, output 4).

Finally, suppose only (c) is false. In this case ADD-1 is working and the value at X is 6 (by assumption), the value at F has been measured to be 10, so we infer that the value at Y must be 4. Hence the third possibility is that MULT-2 is broken (inputs 3 and 2, output 4).

But while that's locally consistent with the value at F, it contradicts the value observed at G. That is, there is no way that MULT-2 alone can malfunction to produce the values observed. We conclude as a result that, given the symptoms shown, the only plausible single-fault candidates are either ADD-1 or MULT-1.

Though this reasoning is very simple, it demonstrates an important principle: knowledge of structure and behavior can form the basis for a powerful kind of diagnostic reasoning. Knowledge of structure is evident in the way in which the reasoning follows the connectivity of the components, moving from F to ADD-1, back through its inputs, etc. Knowledge of behavior was used both for the initial simulation, and later to infer inputs from outputs (e.g., we need to know how an adder behaves to infer the value at X given the values at Y and F).

As it turns out, that principle has some very interesting limitations. Consider Figure 13, with exactly the same circuit, but a new set of symptoms. What can be malfunctioning in this case? The simplest and most direct answer is that the second multiplier is malfunctioning by producing a 0 as its output. The process used just a moment ago will quickly derive this answer.

Interestingly, that's not the only answer. It might also be the case that the third multiplier *alone* might be broken and still account for the symptoms. How could that be?

The answer lies in some interesting assumptions we made about the device. Imagine that MULT-3 is packaged as a single chip all by itself. When chips are plugged into their sockets it sometimes happens that one of the pins doesn't make it into the socket. If that happens to the pin that supplies the power to the chip, then in addition to sending out a zero along its output, the chip can also exhibit the slightly odd behavior of "sending a zero out" along what we thought was going to be an input. More precisely, it can "drag down" its input. (An identical argument can be made for MULT-1 alone as the malfunction.)

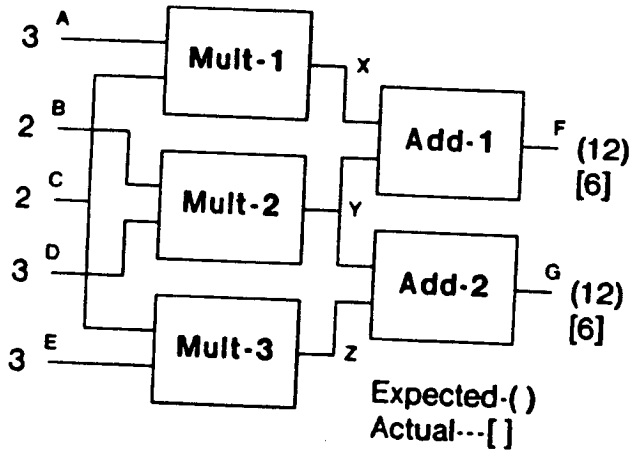


Figure 13. A variant on the first example.

The diagnostic process doesn't seem quite so intuitive anymore. To see one more example of the complexities that can arise, consider Figure 12 once again. Recall that we began the diagnosis by starting at the discrepancy at F and traced back through the connections shown, leading us first to ADD-1, then MULT-1 and MULT-2.

But why do we follow *those* connections? We might for example have said, "perhaps there's another wire, not shown in the diagram, that connects F to the bottom input of ADD-2", and then traced from F down to that input. We don't do that, of course, because the diagrams we are given are always complete and correct.

Or are they? Of course not. The point in making the claim about completeness and correctness so boldly is to make clear how tenuous it really is. There are in fact many ways in which the actual structure of the circuit can differ from the diagram we are given. Errors can occur in assembly: the device may simply be wired up incorrectly. Somewhat more common are problems known as bridging faults. Occasionally, when the chips are soldered into place, instead of small dots of solder at each pin, a

larger puddle forms that is big enough to touch two pins at once, forming a solder "bridge" between them. In effect an extra wire has been inserted between those two pins, a wire that cannot show up on the diagram because it is not supposed to be there.

In the face of this, our original technique of tracing signals through the network seems to be facing serious difficulties. The great virtue of that technique is that it reasons from a knowledge of structure and behavior. But its fatal flaw is that it reasons from a knowledge of structure and behavior. More precisely, it reasons from knowledge of the *intended* structure and behavior, and sometimes the fault lies precisely in the discrepancy between what was intended and what was constructed.

The real question then is not so much how do we trace through the networks we are given, as how do we know what network to trace. That, in turn, seems to open up a seemingly limitless set of difficulties. If we're trying to reason from knowledge of structure and behavior and we can't rely on the description given, what can we do?

We appear to be caught on the horns of a fundamental dilemma: to get anywhere on the problem we need to make *some* assumptions about how the device is structured. Yet any assumption we make could be wrong.

The fundamental dilemma is the tradeoff between complexity and completeness. For the diagnostic routines to be complete, they must consider every possibility, i.e., make no assumptions. But if they do that, they drown in complexity, since they are reduced to exhaustive testing of behavior, which is computationally intractable. To handle the complexity we must make some assumptions, yet that eliminates completeness, since it might cause us to overlook the thing that happens to be wrong.

The problem is broader than electronic troubleshooting alone. Almost any form of real-world problem solving faces this basic issue. To get started on a problem we must make simplifying assumptions; yet to be good at that task we must not be blinded by our assumptions.

The three key ideas that we use to deal with this problem are all quite simple. First, Occam's Razor: start with simple hypotheses and only generate more complex variants as the problem demands it.

Second, make simplifying assumptions, but keep careful track of them and their consequences. When a contradiction arises, the important task is to determine which of the simplifying assumptions was responsible, eliminate that assumption, and try solving the new (slightly more complex) version of the problem.

The final idea is to use multiple representations. In Figure 12 we had a representation of the circuit organized *functionally*, i.e., it shows the various boxes organized according to their purpose in the device. A second way to represent that circuit is shown in Figure 14, where we see it viewed *physically*, as the chips might actually appear on the circuit board. In the case of a bridge fault, this physical representation of the circuit is particularly useful because it helps to suggest where unexpected "wires" might turn up. More generally, we believe, there is for each general category of fault a representation which is particularly useful in just this same sense. In part then, the task of building a troubleshooter for any particular domain involves looking for a set of appropriate representations.

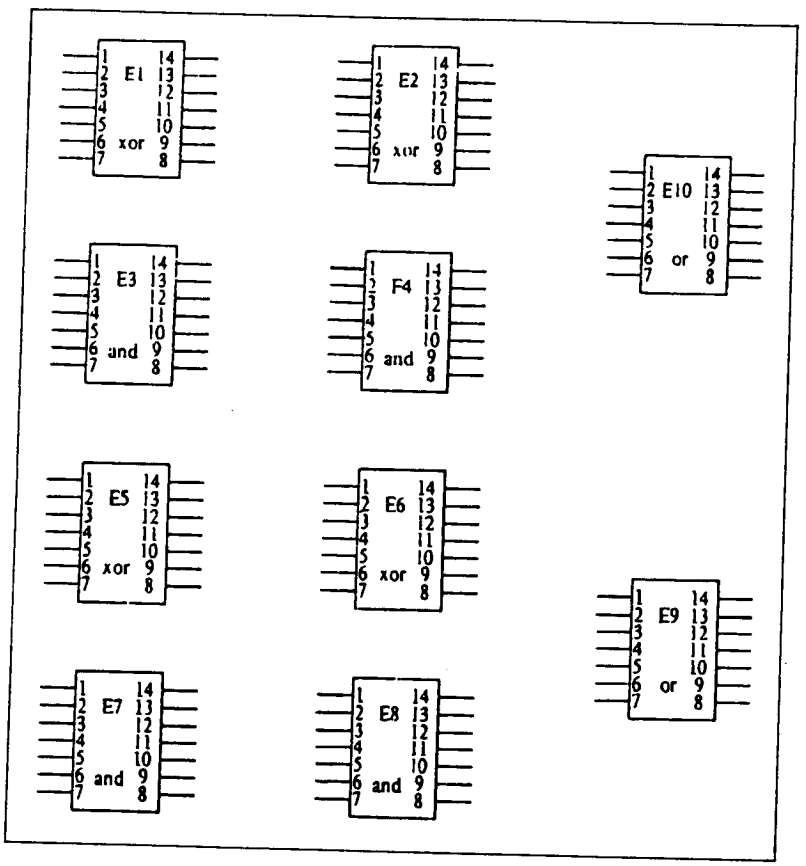


Figure 14. The physical view of the circuit.

These three ideas work together to support a fairly powerful form of diagnostic reasoning. The program can make simplifying assumptions in order to get started, back up when contradictions occur, and then try a slightly more elaborate version of the problem. The use of multiple representations helps by guiding the choice of the more elaborate version of the problem to try next.[6]

Design and Test Programming

The bulk of our experience to date has been on various forms of diagnosis, simulation and test generation [Davis84, Shirley83, Hamscher84, Haig86]. Two other, more recent foci of this line of research are design [Williams85] and test programming [Shirley86]. The fundamental task of design is, given a description of a desired behavior, specify a set of components and their interconnections that will produce that behavior. Test programming involves the creation of a set of tests, used at the end of the manufacturing line, that exhaustively verify the presence of all the intended behaviors of the device.

Current attempts at automated design share two important characteristics: they are largely *library-based* and done by *decomposition*. The basic paradigm of many automated design programs is to look in a library of standard designs to determine what subcomponents are needed, then look up designs for each of those, continuing until primitive components are reached. A design plan for a radio, for example, might decompose the task into designing a power supply, tuner, and audio section, then putting those three together. Given that basic plan, we can now look up a plan for designing each sub-component (i.e., power supply, tuner, audio section).

This can be quite useful and has provided the basis for some programs with interesting behavior. At their best these programs are capable of proposing designs that are novel in the sense that they involve previously unexamined combinations of library plans.

But there are two problems with this. First, there's something missing. There's something unsatisfying about saying that such a program "knows how to design a radio," when in fact all it did was look in a library of pre-existing plans. Second, this approach works with a fixed set of primitives. What happens if we need a tuner that has more sensitivity than is provided by any design plan in the library? All the approach can do is reject the plans

[6] Details of this process are found in the *AI Journal* article referenced earlier.

one after another, until it finally runs out. Since it does not in any sense "understand" any of the designs, it can do nothing to make fundamental changes in any of them.

We would instead like to be able to design something "from first principles", i.e., by understanding how all of its components work and how they contribute to the final behavior. This work, still in its early stages, is founded on several basic ideas [Williams85]:

- The ability of a designer to analyze a complex circuit appears to be based on a core set of very general principles. It may be the case that design, too, is founded on a core set of very general principles, many of which overlap principles used during analysis. Hence a "principled theory of design" involves the identification and use of such principles.
- Second, a theory of design must satisfy both robustness and competence. Robustness means that it is possible to design a wide class of systems based on the theory, while competence means that the design process should be efficient. A core set of design principles derived from analysis will provide robustness: such principles should make it possible to design any of the very large class of systems that is analyzable using the same principles.
- But design using those principles alone is very inefficient. Competence will arise from the use of four ideas: design experience, qualitative abstraction, approximation, and design debugging. Design experience refers to the accumulated experience of a community, captured as a set of design techniques commonly used by skilled designers. These represent shortcuts, well-recognized compositions of basic design principles.
- Qualitative abstraction and approximation allow the designer to focus on exactly those details that are relevant during a particular stage of the design process. These abstractions support a "least commitment" paradigm, enabling the system to delay all design decisions until the latest possible time, reducing the number of commitments that get made early and then later revised.
- Finally, the notion of design and debug suggests that it is at times considerably more enlightening to get the design "mostly correct" and then try it out, than to attempt to anticipate every detail. The system will recognize how and why a previous commitment turned out to be incorrect, then redesign based on what has been noted.

The task in test programming is not diagnosis of devices that have been working and suddenly begin to exhibit symptoms of misbehavior, but rather the complete verification of performance for a newly constructed device. The problem is particularly important in VLSI design because chip production has in recent years become increasingly test-limited. Where it was originally limited by problems in fabrication, and later by problems in complexity in design, it is more recently the case that we can design and fabricate chips complex enough that it is extraordinarily difficult to determine whether they actually work. Devices are becoming sufficiently complicated that none of the existing techniques for automated design of test programs is good enough. On complex chips existing test generation programs may never stop, running for weeks without producing an answer.

Interestingly, when this happens, chip manufacturers turn to human experts capable of solving the problem. Their skill in turn appears to rely on several interesting problem solving methods [Shirley86]. First, the experts remember many clichéd solutions to test programming problems. The difficulty in getting a program to this lies in formalizing the notion of a cliché for this domain. For test programming the clichés appear to contain fragments of test program code and constraints describing how program fragments fit together.

Second, experts can simulate a circuit at various levels of abstraction and recognize patterns of activity in the circuit that are useful for solving testing problems. An effective planning strategy can be based on symbolic simulation, coupled with recognition of the simulated events that accomplish the goal.

Third, and perhaps most important, this planning is guided by the knowledge that the device was purposefully designed to perform a set of specific tasks. The key here is that a device's *designed* behavior is far more limited than its *potential* behavior. This limitation translates into a reduction of the search necessary to achieve goals.

In addition to troubleshooting, design, and test program generation, we are also exploring tasks like design for testability. This is in some ways the dual problem to generating test programs: we can either design freely and then work hard to find a set of tests, or we can design with testability in mind and make the test generation task simpler. A wide range of applicable design techniques have been assembled over the years, but their use is for the most part a rather subtle art. We are attempting to codify some of that art and use it to build a system that will eventually be capable of

examining a proposed design and suggesting modifications only where the circuit is not currently testable.

Summary

In reviewing current technology we saw that rules and frames are two of the most common and best understood approaches to building knowledge-based systems. Rules capture informal, often judgemental heuristics, and are typically chained together, to form a line of reasoning leading to the answer. Backward chaining (from goals back toward more primitive data) is widely used because it mimics human problem solving; rules can also be run forward, making all logical conclusions from a basic set of input data. Frames capture prototypical situations and reason primarily by matching those prototypes against specific instances.

In examining today's commercial environment, we see a change in the kinds of systems being built. The original applications were all tools designed to solve in-house problems in production, quality, etc., all assisting in doing the existing business faster, better, or cheaper. More recently we are seeing the technology used to produce new products enabling strategic moves by organizations interested in new markets, and seeing products that are much more responsive to existing market needs, more market-driven than technology-driven.

Finally, we reviewed briefly some of the research currently underway at MIT, exploring the theme of reasoning from first principles, a theme we believe will be at the core of the next generation of knowledge-based systems. Its significance lies in part in its ability to provide an additional level of understanding, and as a result, an additional level of power in a whole range of tasks, including maintenance, test generation, circuit design and so forth. Our current work is aimed at digital electronic circuits because of the simplicity of the basic components, but we believe that the underlying ideas developed are applicable across a broad range of domains.

References

Davis, R., Buchanan, B., Shortliffe, E., 1977, " Production rules as a representation in a knowledge-based consultation system," *Artificial Intelligence*, 18, pp. 15–45.

Davis, R., 1984, " Diagnostic reasoning from structure and behavior," *Artificial Intelligence*, pp. 347–410.

Etherington, W., Reiter, W., "On inheritance hierarchies with exceptions," *Proc AAAI-83*, Washington, DC., pp. 104–108.

Fikes, R., Kehler, T., 1985, "The role of frame-based representation in reasoning," *Comm ACM*, pp. 904–920.

Haig, H., 1986, A language for describing digital circuits, MS Thesis, MIT, Computer Science.

Hamscher, W., Davis, R., 1984, " Diagnosing circuits with state, an inherently underconstrained problem," *Proc AAAI-84*, Austin, Texas, pp. 142–147.

Minsky, M., 1975, "A framework for representing knowledge," *The Psychology of Computer Vision*, edited by P. H. Winston, McGraw-Hill, pp. 211–277.

Pople, H., 1982, "Heuristic methods for imposing structure on ill-structured problems," *Artificial Intelligence in Medicine*, edited by P. Szolovits, Westview Press, AAAS Symposium Series, pp. 119–190.

Shirley, M., Davis, R., Oct 1983, "Generating distinguishing tests from hierarchical models and symptom information," *Proc IEEE International Conference on Computer Design.*

Shirley, M., "Generating tests by exploiting designed behavior," *Proc AAAI-86.*

Shortliffe, E., Buchanan, B., 1975, "A model of inexact reasoning in medicine," *Mathematical Biosciences*, 23, pp. 351–379.

Williams, B., 1985, Principle designed based on qualitative behavioral descriptions, MS Thesis, MIT, Computer Science.

EXPERT SYSTEMS

TOOLS & APPLICATIONS

Paul Harmon Rex Maus
William Morrissey

The figures appearing in this book, unless otherwise noted, originally appeared in the *Expert Systems Strategies* newsletter and are reproduced with the permission of Harmon Associates, San Francisco, CA.

Figures 3.1, 4.1, and 10.2, and Tables 2.5 and 3.1 are modifications of figures and tables that originally appeared in *Expert Systems: Artificial Intelligence in Business*, by Paul Harmon and David King (John Wiley & Sons, 1985).

Publisher: Stephen Kippur
Editor: Therese A. Zak
Managing Editor: Ruth Greif
Editing, Design & Production: G&H SOHO, Ltd.

Library of Congress Cataloging-in-Publication Data

Harmon, Paul.
Expert systems tools and applications /
Paul Harmon, Rex Maus, William Morrissey.
p. cm.
1. Business—Data processing. 2. Expert systems (Computer science) I. Maus, Rex. II. Morrissey, William. III. Title.
HF5548.2.H367 1988
650'.028'5633—dc19 87-17608
CIP

ISBN 0-471-83951-5
ISBN 0-471-83950-7 (pbk.)

Printed in the United States of America

88 89 10 9 8 7 6 5 4 3 2 1

haustive search can become impossible. Methods are domain independent strategies like "generate and test." Strong methods exploit domain knowledge to achieve greater performance. This is usually accomplished by avoiding exhaustive search in favor of exploring a few likely solutions.

Problem Space. A conceptual or formal area defined by all of the possible states that could occur as a result of interactions between the elements and operators that are considered when a particular problem is being studied.

Procedural versus Declarative. Procedures tell a system what to do (e.g., Multiply A times B and then add C). Declarations tell a system what to know (e.g., V=IR).

Programming Environment (Environment). A programming environment is about halfway between a language and a tool. A language allows the user complete flexibility. A tool constrains the user in many ways. A programming environment, like OPS5, provides a number of established routines that can facilitate the quick development of rule-based programs.

PROLOG. A symbolic or AI programming language based on Predicate Calculus. PROLOG is the most popular language for AI research outside North America.

Prototype. In expert systems development, a prototype is an initial version of an expert system that is developed to test effectiveness of the overall knowledge representation and inference strategies being employed to solve a particular problem.

Pruning. In expert systems, this refers to the process whereby one or more branches of a decision tree are "cut off" or ignored. In effect, when an expert systems consultation is under way, heuristic rules reduce the search space by determining that certain branches (or subsets of rules) can be ignored.

Reasoning. The process of drawing inferences or conclusions.

Representation. The way in which a system stores knowledge about a domain. Knowledge consists of facts and the relationships between facts. Facts, rules, objects, and networks are all formats for representing knowledge.

Robotics. The branch of AI research that is concerned with enabling computers to "see" and "manipulate" objects in their surrounding environment. AI is not concerned with robotics as such, but it is concerned with developing the techniques necessary to develop robots that can use heuristics to function in a highly flexible manner while interacting with a constantly changing environment.

Rule (If-Then Rule). A conditional statement of two parts. The first part, composed of one or more if clauses, establishes conditions that must apply if a second part, composed of one or more than clauses, is to be acted upon. The clauses of rules are usually A–V pairs or O–A–V triplets.

Rule-Based Program. A computer program that represents knowledge by means of rules.

Runtime Version or System. Knowledge system building tools allow the user to create and run various knowledge bases. Using a single tool, a user might create a dozen knowledge bases. Depending on the problem the user was facing, he or she would load an appropriate knowledge base and undertake a consultation. With such a tool the user can easily modify a knowledge base. Some companies will want to develop a specific knowledge base and then produce copies of the tool and that specific knowledge base. Under these circumstances the organization will not want the user to have to "load" the knowledge base, nor will they want the user to be able to modify the knowledge base. When an expert system building tool is modified to incorporate a specific knowledge base and to deactivate certain programming features, the resulting system is called a runtime system or a runtime version.

Search and **Search Space.** See Problem-Solving and Problem Space.

Semantic. Refers to the meaning of an expression. It is often contrasted with syntactic, which refers to the formal pattern of the expression. Computers are good at establishing that the correct syntax is being

Teresa M. Corbin (SBN 132360)
Thomas C. Mavrakakis (SBN 147674)
HOWREY SIMON ARNOLD & WHITE, LLP
301 Ravenswood Avenue
Menlo Park, California 94025
Telephone: (650) 463-8100
Facsimile: (650) 463-8400

Attorneys for Synopsys, Inc. and
Aeroflex, Inc., et al.

UNITED STATES DISTRICT COURT

NORTHERN DISTRICT OF CALIFORNIA

RICOH COMPANY LTD., Plaintiff, v. AEROFLEX, INC., ET AL., Defendant.	Case No. C-03-4669 MJJ (EMC) Case No. C-03-2289 MJJ (EMC) **PROOF OF SERVICE OF EXHIBITS 4 AND 15 TO RESPONSIVE CLAIM CONSTRUCTION BRIEF FOR U.S. PATENT NO. 4,922,432**
SYNOPSYS, INC., Plaintiff, v. RICOH COMPANY, LTD., Defendant.	

STATE OF CALIFORNIA) ss.:
)
COUNTY OF SAN MATEO)

I am employed in the County of San Mateo, State of California. I am over the age of 18 and not a party to the within action. My business address is 301 Ravenswood Avenue, Menlo Park, CA 94025.

On September 10, 2004, I served the within:

EXHIBITS 4 AND 15 TO RESPONSIVE CLAIM CONSTRUCTION BRIEF FOR U.S. PATENT NO. 4,922,432

by placing true copies thereof in a sealed envelope(s) addressed as stated below and causing such envelope(s) to be delivered via:

☒ (PERSONAL SERVICE) I caused each such envelope to be delivered by hand to the offices of the interested party below.

Jeffrey B. Demain
Jonathan Weissglass
Altshuler, Berzon, Nussbaum, Rubin & Demain
177 Post Street, Suite 300
San Francisco, CA 94108

☒ (OVERNIGHT DELIVERY) by depositing in a box or other facility regularly maintained by Federal Express, an express service carrier, or delivering to a courier or driver authorized by said express service carrier to receive documents, a true copy of the foregoing document in sealed envelopes or packages designated by the express service carrier, addressed as stated below, with fees for overnight delivery paid or provided for and causing such envelope(s) to be delivered by said express service carrier.

Edward A. Meilman
Dickstein Shapiro Morin & Oshinsky, LLP
1177 Avenue of the Americas
New York, NY 10036

Gary M. Hoffman
Dickstein Shapiro Morin & Oshinsky, LLP
2101 L Street NW
Washington, DC 20037

I declare under penalty of perjury that I am employed in the office of a member of the bar of this Court at whose direction the service was made and that the foregoing is true and correct.

Executed on September 10, 2004, at Meno Park, California.

Susan J. Crane
(Type or print name)

/s/
(Signature)